Wright State University

Report-P5

Paul Fuchs

U00747698

CEG 3900 Mobile and Cloud Computing

Prabhaker Mateti

26 March 2017

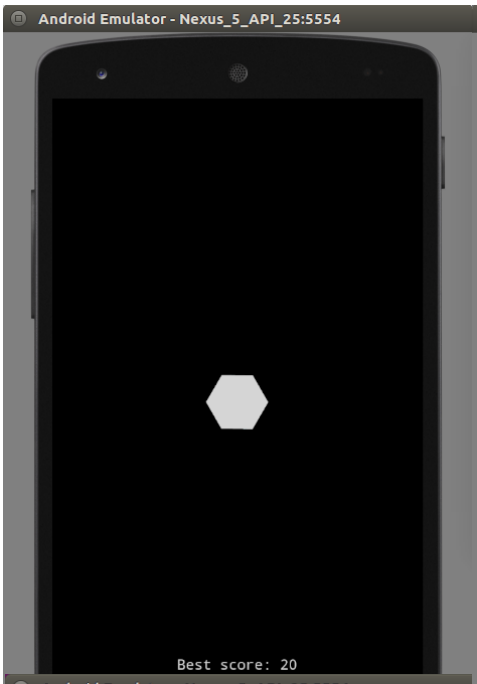
https://github.com/helmhammerhand/ceg-3900-spring-2017/

Table of Contents

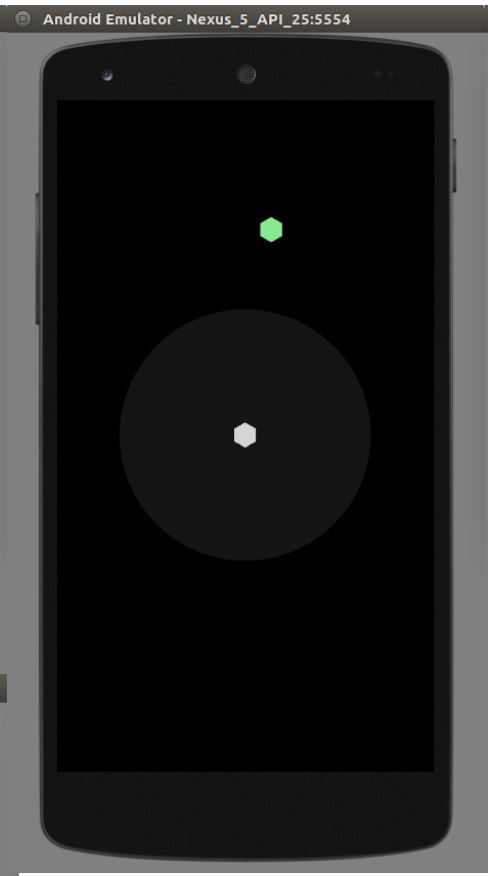
3.1 Program Comprehension (1 hour)

In order to compile this APK, I had to install the Kotlin plugin for AndroidStudio. After this, I could compile and run Uniting Twist. I did this on the emulator, so the results were not too interesting as I could not use a gyroscope. I did manage to get a few points by lettign the hexagons fall without twisting the phone. Below are screenshots fo the running app.

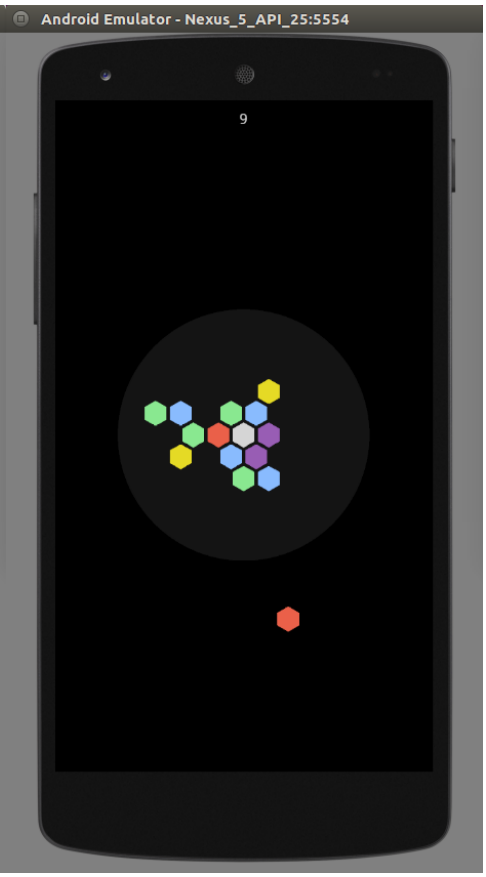
Starting Screen



Beginning of Game



Middle of Game



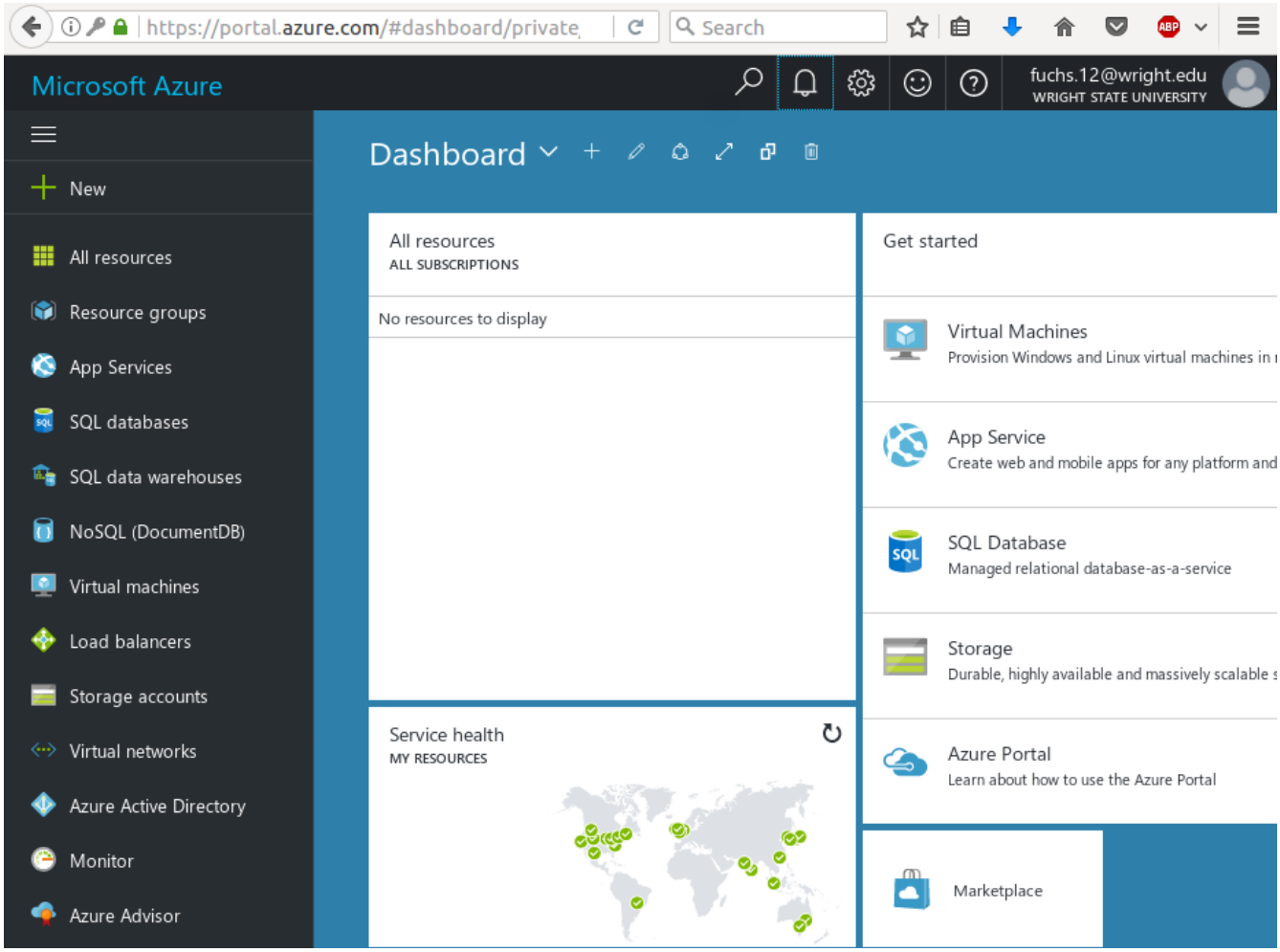
<- Game Over

I started but did not complete the program comprehension due to a lack of time.

3.2 MS Azure Photo Uploader (1.5 hours)

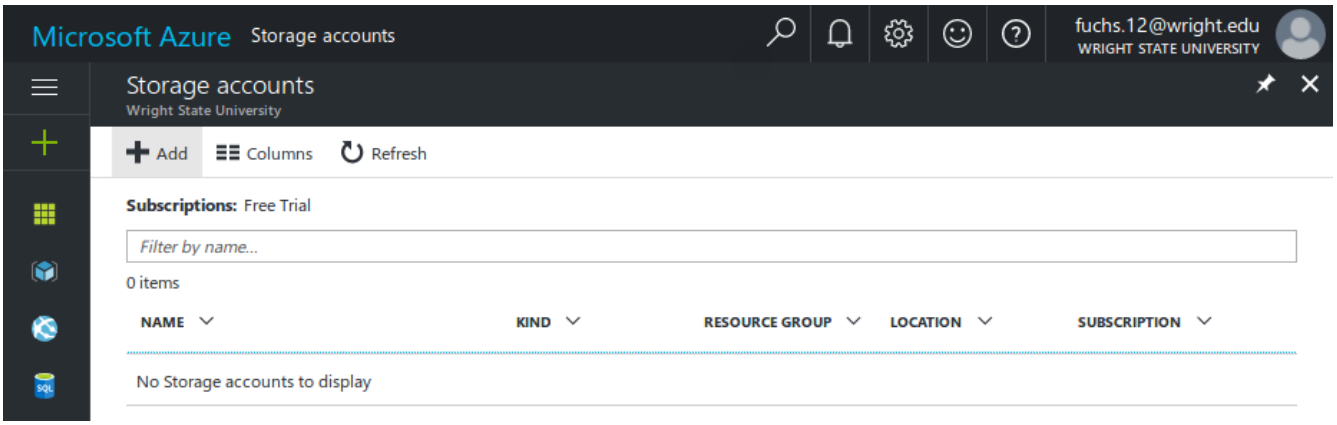
Setting up Azure

Logging into Azure by creating a free account brought up the dashboard.

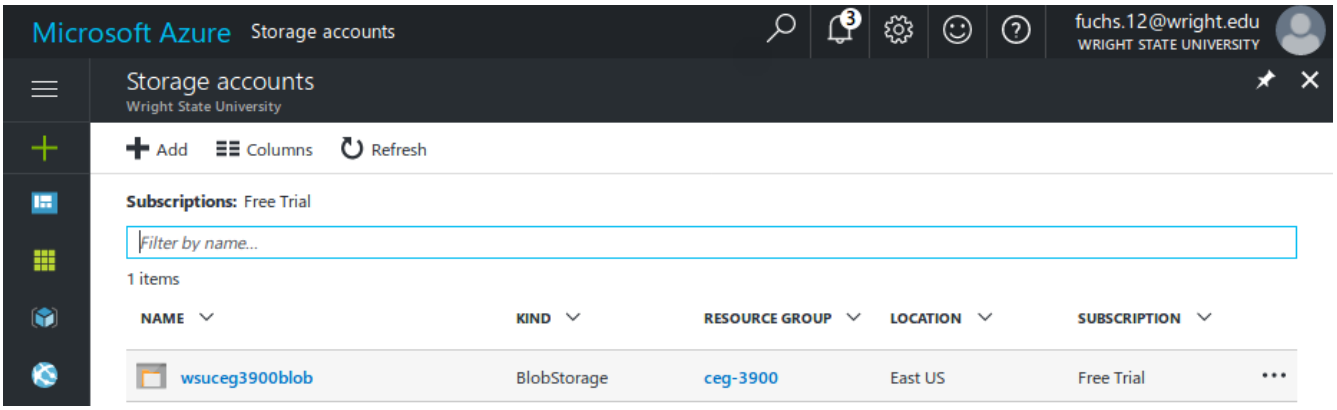


After logging in, I added a storage Blob as shown on the next page.

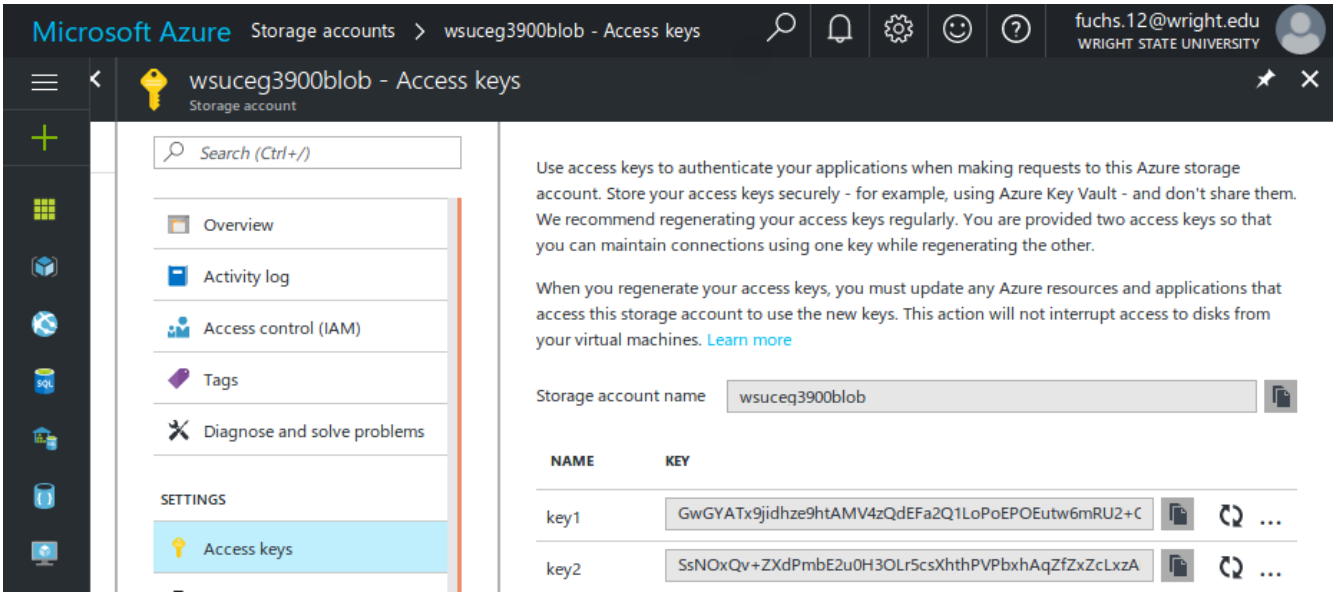
Under ‘Storage accounts’ I selected the add button to create a blob.



I named the blob ‘wsuceg3900blob’ as shown below



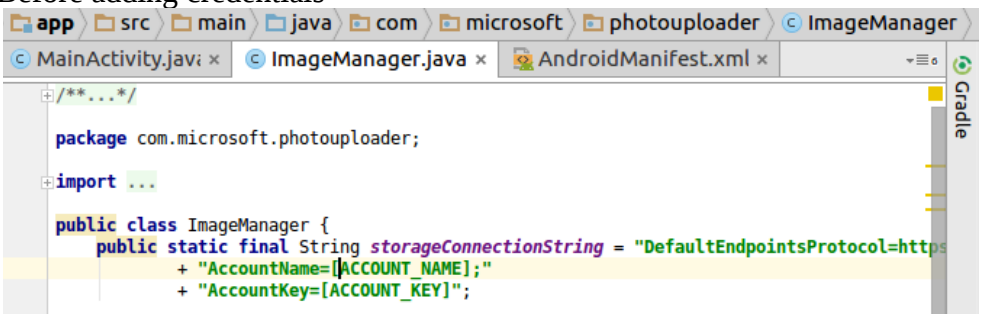
The credentials were found under the access keys menu under storage account



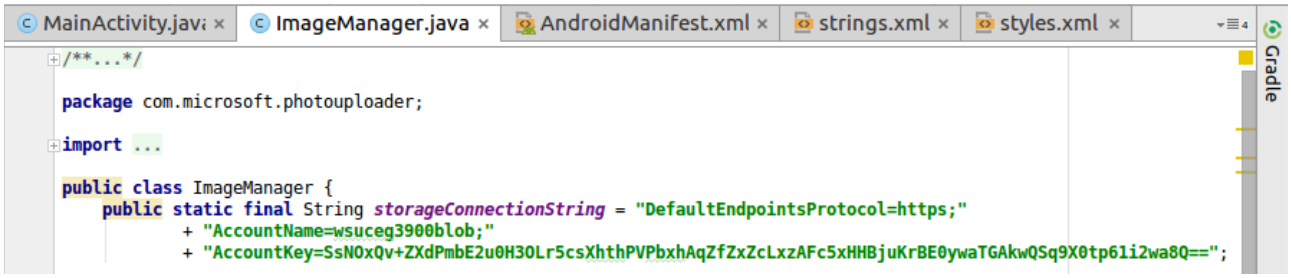
Building the APK

I changed build version numbers to match the Gradle and Android SDK versions installed on my computer. I also entered the account name and account key for my blob storage into ImageManager.classas shown below.

Before adding credentials

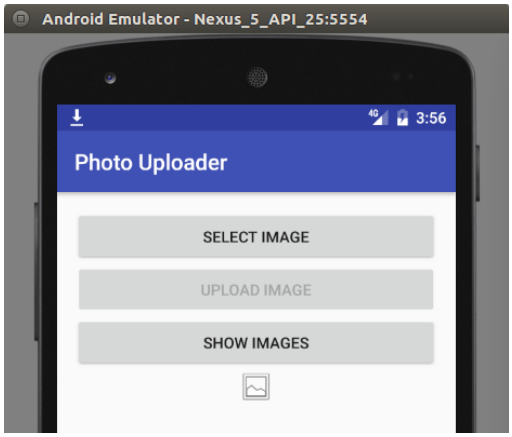


With credentials

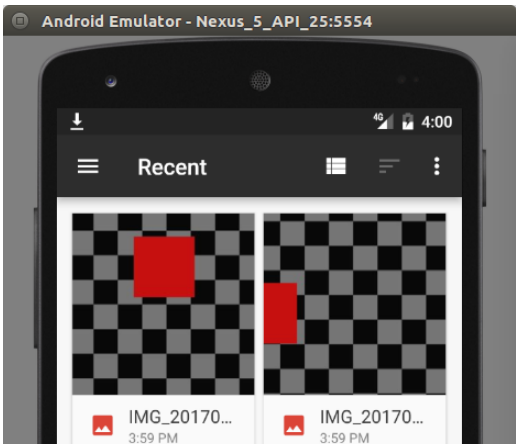


Screenshots

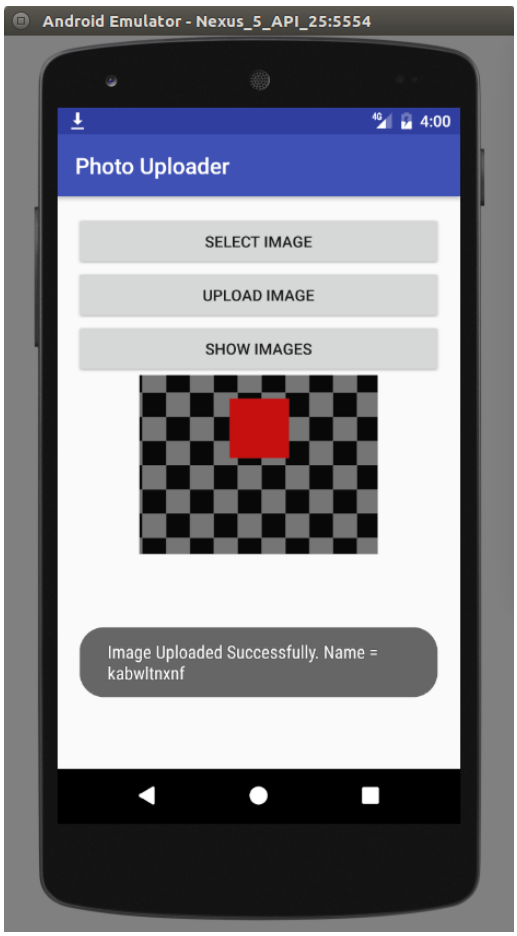
Initial App Screen



Selecting an Image

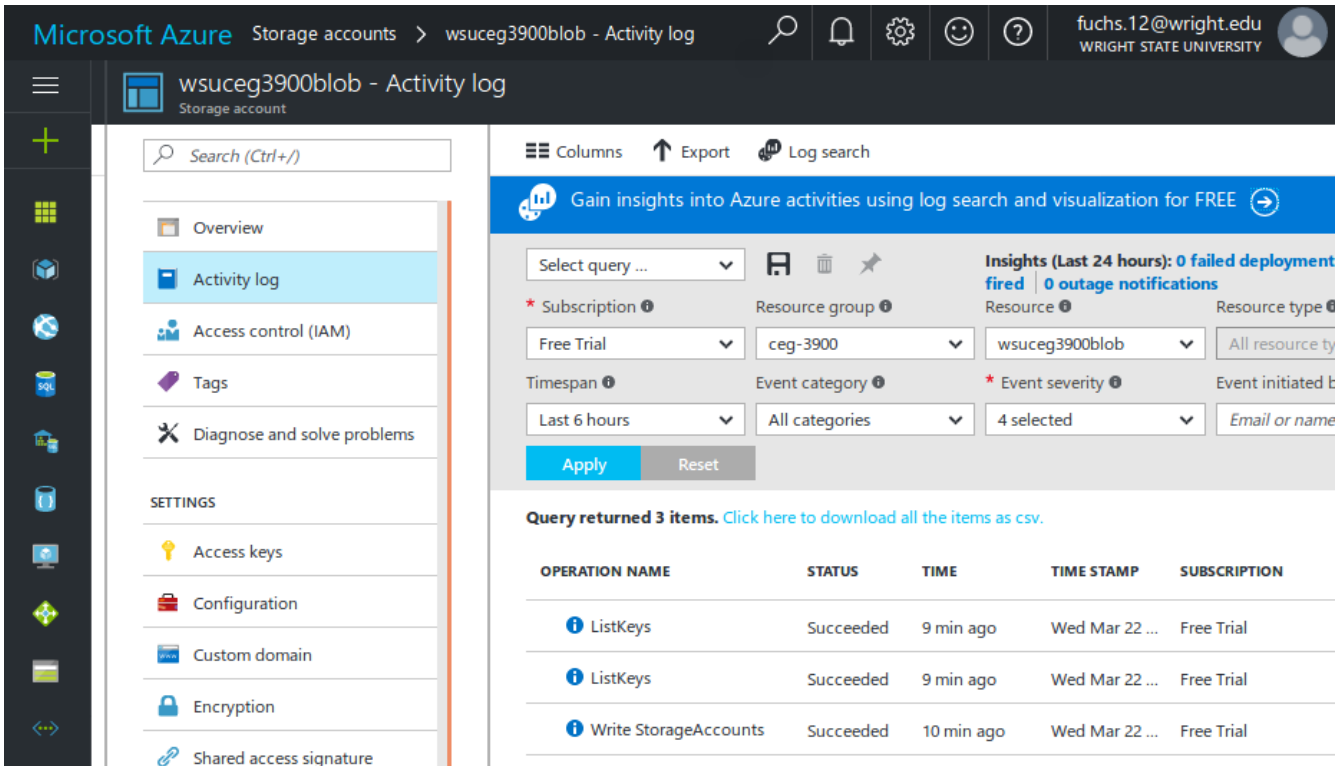


Uploading an Image



Show Images – displays names of all images that have been uploaded

Azure dashboard showing image read and write events



3.3 Google Cloud Platform Bookshelf (7 hours)

Running the Bookshelf App Using Maven (4 hours)

Google cloud SDK

The tutorial contained a link to download the google cloud sdk. I downloaded and installed it by running the installer script. The SDK is utilized via the gcloud command shown below.

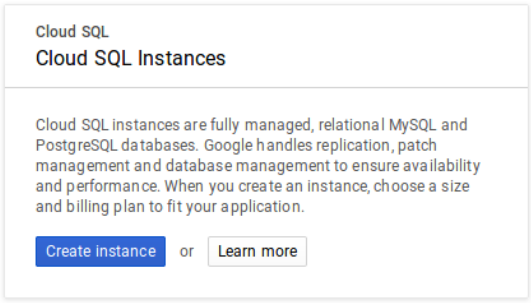
```
frodo@24601:~/CEG 3900$ gcloud
ERROR: (gcloud) too few arguments
Usage: gcloud [optional flags] <group | command>
  group may be           app | auth | components | compute | config |
                         container | dataflow | dataproc | datastore | debug |
                         deployment-manager | dns | iam | kms | ml-engine |
                         organizations | projects | service-management |
                         source | sql | topic
  command may be         docker | feedback | help | info | init | version

For detailed information on this command and its flags, run:
  gcloud --help
```

Installing Java app engine on gcloud. In order to run the Java example program, I had to install a plugin for gcloud as shown below.

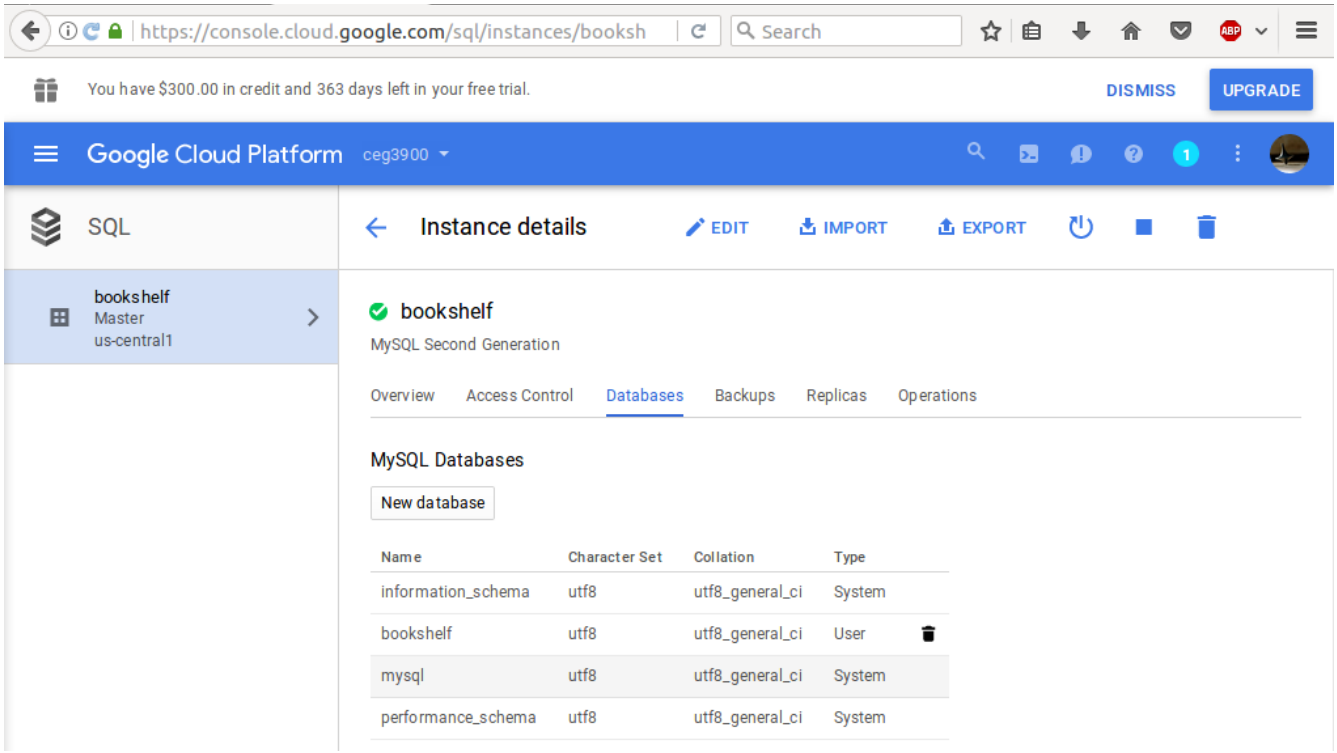
```
frodo@24601: ~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data
Java version: 1.8.0_121, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.4.0-66-generic", arch: "amd64", family: "unix"
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
gcloud components install app-engine-java


Your current Cloud SDK version is: 148.0.0
Installing components from version: 148.0.0

┌──────────────────────────────────────────────────────┐
│          These components will be installed.         │
├──────────────────────────────┬──────────┬────────────┤
│             Name             │ Version  │    Size    │
├──────────────────────────────┼──────────┼────────────┤
│ gcloud app Java Extensions   │  1.9.50  │ 128.6 MiB  │
│ gcloud app Python Extensions │  1.9.50  │   7.2 MiB  │
└──────────────────────────────┴──────────┴────────────┘


For the latest full release notes, please visit:
  https://cloud.google.com/sdk/release_notes

Do you want to continue (Y/n)?
```

Google cloud database

Next, I had to create a database on the cloud for the app to store things in. Under the databases item, I selected Create Instance and set up the database type and information such as username and password.



After creation, I had to add a table named ‘bookshelf’ since that was the name specified in the pom.xml file. Below shows the details of my SQL database.

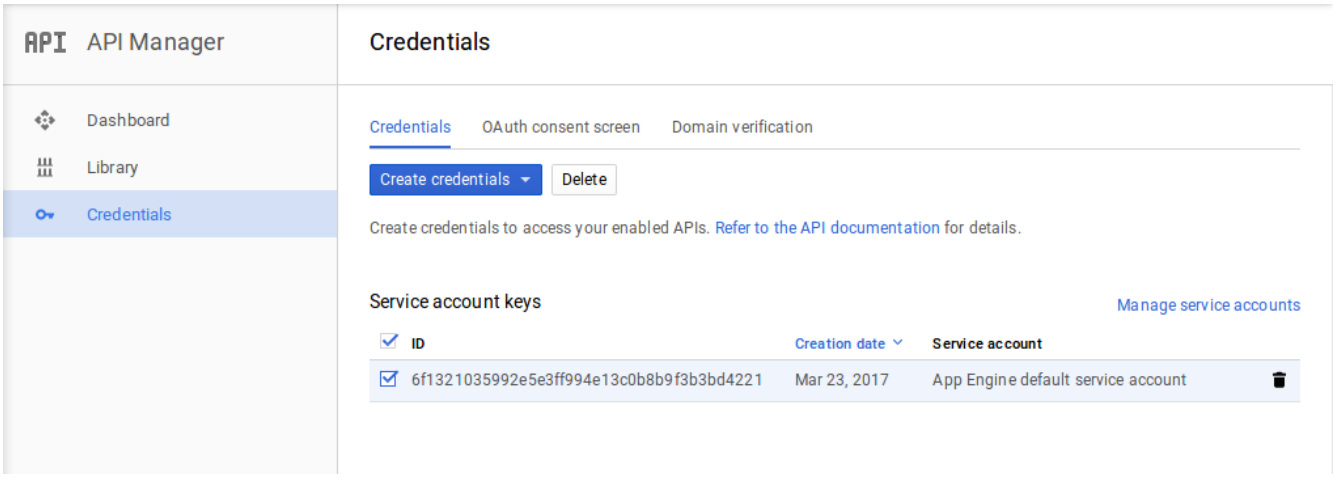


At this point I realized that the pom file specfied bookshelf as a datastore, not SQL database. To get around this, I changed the storage type in the pom file from datastore to cloudsql as shown below. I also entered my database username and password under the appropriate database tags

```
    <version>1.0-SNAPSHOT</version>
    <relativePath>../</relativePath>
  </parent>

  <properties>
    <!-- [START config] -->
    <projectID>ceg3900-162320</projectID> <!-- set w/ -DprojectID=myProjectID on command line -->

    <bookshelf.storageType>cloudsql</bookshelf.storageType>   <!-- datastore or cloudsql -->

    <sql.dbName>bookshelf</sql.dbName>                        <!-- A reasonable default -->
<!-- Instance Connection Name - project:region:dbName -->
<!-- -Dsql.instanceName=localhost to use a local MySQL server -->
    <sql.instanceName>${projectID}:us-central1:${sql.dbName}</sql.instanceName>
    <sql.userName>root</sql.userName>                         <!-- A reasonable default -->
    <sql.password>letmein1234</sql.password> <!-- -Dsql.password=myRootPassword1234 -->
    <!-- [END config] -->
```

Creating auth credentials

https://developers.google.com/identity/protocols/application-default-credentials
In order for the app to work, I had to download the credentials and set the environment GOOGLE_APPLICATION_CREDENTIALS to point to the file containing them.



Setting the environment variable

```
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
export GOOGLE_APPLICATION_CREDENTIALS=~/CEG\ 3900/getting-started-java-master/ce
g3900-6f1321035992.json
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
echo $GOOGLE_APPLICATION_CREDENTIALS
/home/frodo/CEG 3900/getting-started-java-master/ceg3900-6f1321035992.json
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
```

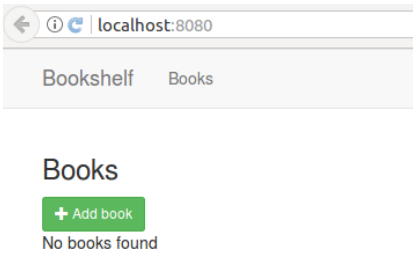
Running the app

First, I had to install maven which I did using apt. Then I used the below command from the tutorial to start the web application which ran on port 8080 of my localhost.

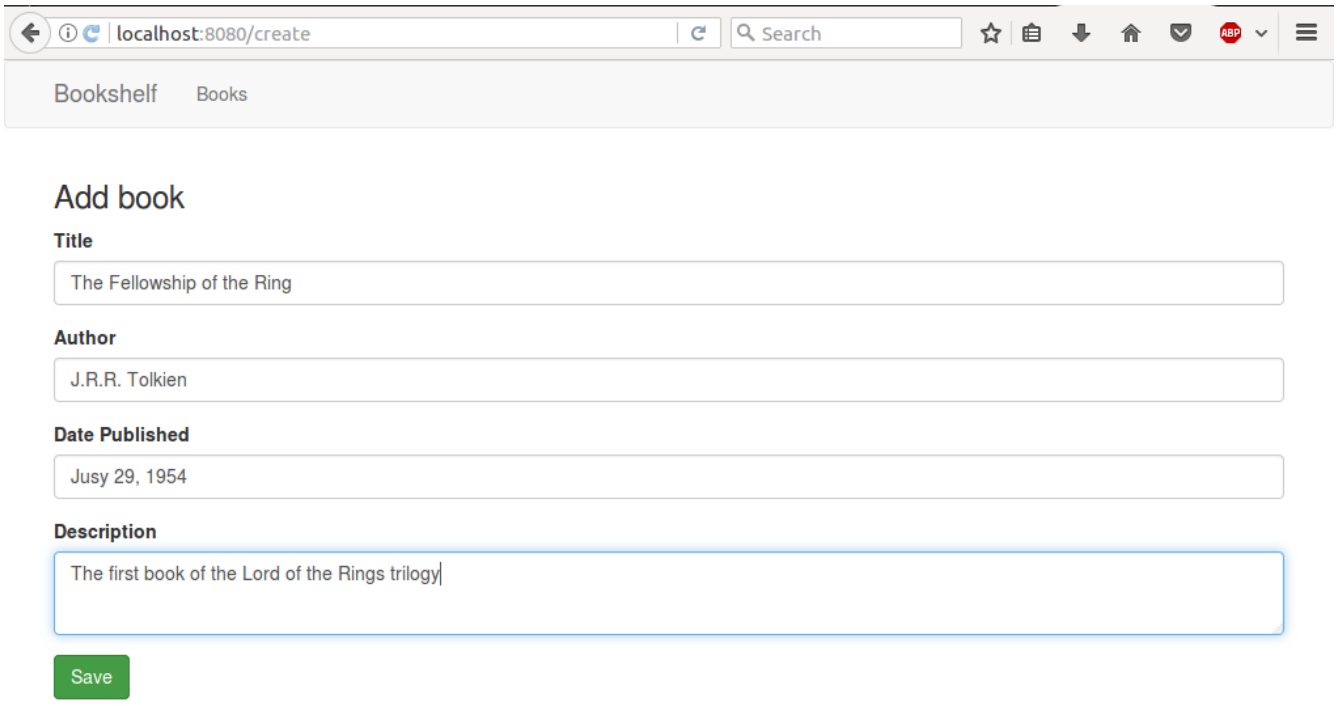
This task took much longer than necessary because I did these steps in the wrong order. At first, I did not create a GCP database which caused the webapp to generate errors. After fixing that, I discovered that I needed to install the app engine extension for the google cloud SDK. Finally, I did not have the authentication credentials properly set up as they were not explicitly explained in the tutorial.

Screenshots

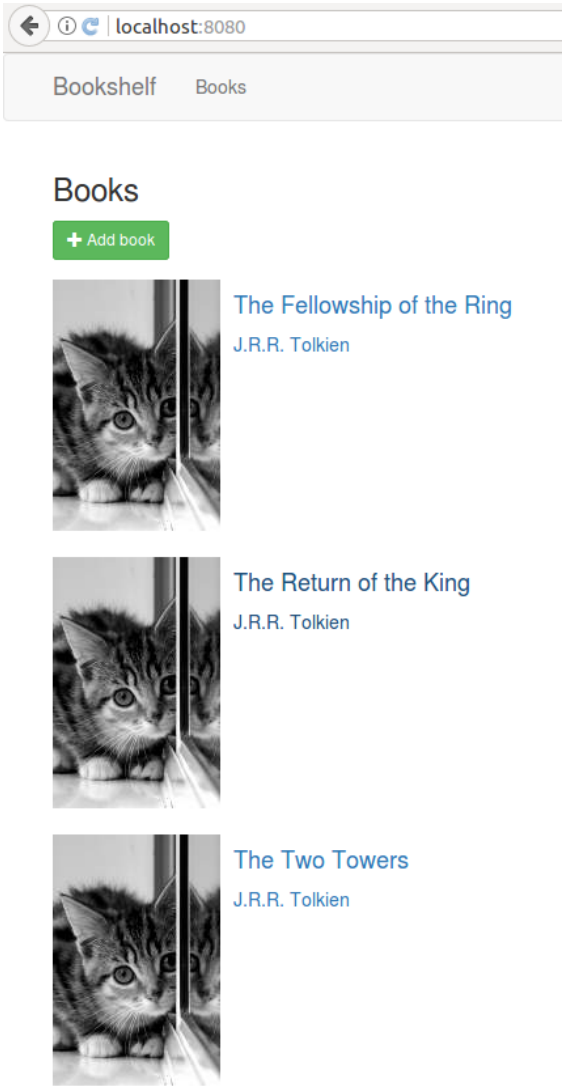
Initial Screen



Selecting Add book brings up this dialog



After adding several books



Selecting a book shows its information

Gradle Conversion (3 hours)

Initially, I installed Gradle using ‘sudo apt-get install gradle,’ but this installed gradle version 2.10.0 This lead to an cyclic graph error when trying to execute ‘gradle init’. I tried removing different sections of the pom.xml but could not get ‘gradle init’ to work. It kept giving a cyclic graph error having something to do with the maven plugin manager.

Finally, I realized gradle was the wrong version and manually downloaded and installed gradle 3.4.1 following the instructions from https://gradle.org/install. After this, ‘gradle init’ converted the maven files to gradle without a problem.

At this point, I realized that I could run the app using Maven as outlined in the tutorial. Additionally, to turn this into and APK would require either running the bookshelf Java app on a server somewhere and implementing an Android client or modifying all the networking code in the example to use the Android libraries. As I already spent more time than I would have liked on this, I decided not to move forewards with the android APK.

Initial Build Failure Error

```
Caused by: org.codehaus.plexus.util.dag.CycleDetectedException: Edge between 'Ve
rtex{label='org.apache.maven.plugin.MavenPluginManager:default'}' and 'Vertex{la
bel='org.apache.maven.plugin.version.PluginVersionResolver:default'}' introduces
 to cycle in the graph org.apache.maven.plugin.version.PluginVersionResolver:def
ault --> org.apache.maven.plugin.MavenPluginManager:default --> org.apache.maven
.plugin.version.PluginVersionResolver:default
        at org.codehaus.plexus.util.dag.DAG.addEdge(DAG.java:143)
        at org.codehaus.plexus.util.dag.DAG.addEdge(DAG.java:123)
        at org.codehaus.plexus.component.composition.DefaultCompositionResolver.
addComponentDescriptor(DefaultCompositionResolver.java:60)
        ... 79 more


BUILD FAILED
```

Wrong Gradle Version

```
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
gradle -v

------------------------------------------------------------
Gradle 2.10
------------------------------------------------------------
```

Successful Build!

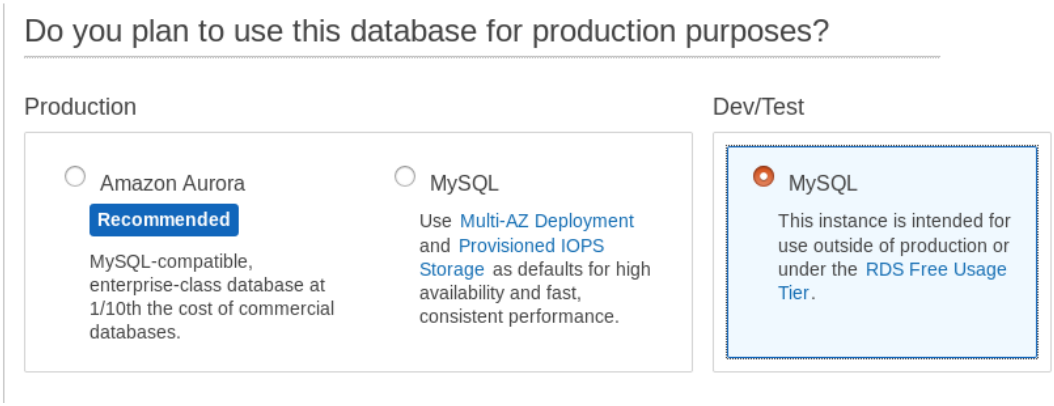
```
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
/opt/gradle/gradle-3.4.1/bin/gradle init
:wrapper UP-TO-DATE
:init
Maven to Gradle conversion is an incubating feature.

BUILD SUCCESSFUL

Total time: 0.855 secs
frodo@24601:~/CEG 3900/getting-started-java-master/bookshelf/2-structured-data$
```

3.4 Collaborative Tagging of XKCD Cartoons (10 hours)

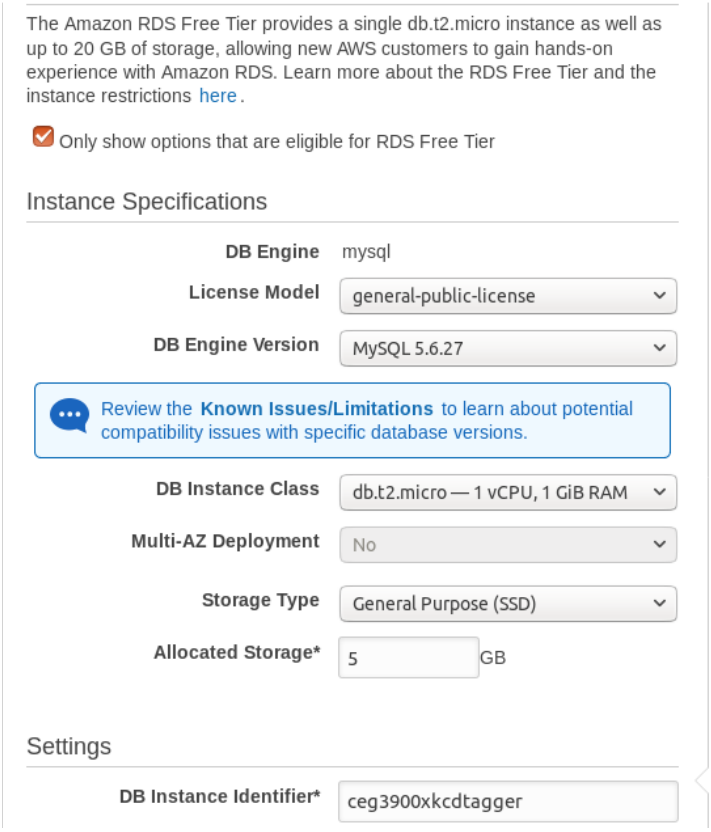
This is the first time I have written much PHP or SQL so this took me a while.

Amazon RDS (1 hour)

For this task, I used an Amazon RDS (Relational Database Service) SQL database for storing information about tags. Below are screenshots showing the creation and configuration of the database.



Database Configuration



The database was set up as described below:

Instance identifier: ceg3900xkcdtagger
Database namer: xkcdtags
Tables:
- Tags: tag_id, name
- TagMap: id, tag_id, comic_id
- Comics: id, name, url

I used a 3 table database achitecture based on ‘Toxi’ solution from http://tagging.pui.ch/post/37027745720/tags-database-schemas. One table stores the comic data (Comics), one the tag data (Tags), and the other links tags to comics (TagMap).

The id numbers are automatically generated and store the row of each entry in the table. The tag names are provided by the user. Comic names store the comic number (not the title) because it is easily accessible in the EasyXKCD apk and uniquely identifies each comic.

EC2 PHP Server (7 hours)

The database front end is a web server running on an EC2 istance. I used about 200 lines of PHP to perform all the database operations and send responses back to a client I implemented on the XKCD viewer.

I installed XAMPP web server to use for my server webpage. XAMPP was quite easy to use since all I had to do was place my index.php file in /oct/lampp/htdocs and start the server. I could then use a browser to test the server. Data is passed to index.php using HTTP GET parameters as outlined in the next section.

Server Usage

Based on the description provided in P5, the server needs five functions. Below are the get parameters for and a brief description of each function. The words in CAPS are parameters which will be sent by the app.

Add tag to comic:	/index.php?type=tag&comic=COMIC_NAME&tag=TAG_NAME
Remove tag from comic:	/index.php?type=untag&comic=COMIC_NAME&tag=TAG_NAME
List all tags in database:	/index.php?type=taglist (not necessary but makes usage easier)
List all tags for a comic:	/index.php?type=comictags&comic=COMIC_NAME
List all comics with a tag:	/index.php?type=searchtag&tag=TAG_NAME

SQL Queries

Below are the SQL queries used to accomplish each server function. All parameters were passed via $_GET[] as outlined above. Thus, in the below queries TAG_NAME is implemented as escape_string($_GET[“tag”]) and COMIC_NAME is implemented as escape_string($_GET[“comic”]). A switch statement based on $_GET[“type”] is used to select which query is made.

Add tag TAG_NAME to comic COMIC_NAME

Add tag first checks that both the tag and comic are in the Tags and Comics tables respectively. If the comic or tag is not already in the database, it is added. This is necessary so users can define tags or add comics as more are published. Finally, an entry is added to TagMap to associate the comic with the tag.

```
//Check if COMIC_NAME is in Comics (if query returns 1 row)
SELECT DISTINCT id FROM Comics WHERE name = COMIC_NAME

//If no rows are returned, the comic is added to Comics using
INSERT INTO Comics (name) VALUE (COMIC_NAME)

//Check if TAG_NAME is in Tags (if query returns 1 row)
SELECT DISTINCT `tag_id` FROM `Tags` WHERE `name` = 'TAG_NAME'

//If no rows are returned, the tag is added to Tags using
INSERT INTO Tags (name) VALUE (TAG_NAME)

//Adding the comic and tag pair to TagMap. COMIC_ID and TAG_ID are found using the SELECT
//DISTINCT queries shown above.
INSERT INTO `TagMap` (comic_id, tag_id)
VALUES (‘COMIC_ID’, ‘TAG_ID’)
```

Remove tag TAG_NAME from comic COMIC_NAME

This is done in several steps to ensure that the specified tag and comic actually exist in the database.

```
//Get COMIC_ID, the row of the comic in Comics (does not exist if no rows are returned)
SELECT DISTINCT id FROM Comics WHERE name = COMIC_NAME

//Get TAG_ID, the row of the tag in Tags (does not exist if no rows are returned)
SELECT DISTINCT `tag_id` FROM `Tags` WHERE `name` = 'TAG_NAME'

//Disassociate the comic and tag by removing from TagMap
DELETE FROM TagMap
WHERE comic_id = ‘COMIC_ID'
AND tag_id = 'TAG_ID'
```

List all tags in Tags

```
SELECT `name` FROM `Tags`
```

List all tags TAG_NAME for a comic COMIC_NAME

```
SELECT t.name FROM Tags t, Comics c, TagMap tm
WHERE t.tag_id = tm.tag_id
AND c.id = tm.comic_id
AND c.name = ‘COMIC_NAME’
```

List all comics COMIC_NAME with tat TAG_NAME

```
SELECT c.name FROM Tags t, Comics c, TagMap tm
WHERE t.tag_id = tm.tag_id
AND c.id = tm.comic_id
AND t.name = 'TAG_NAME’
```

server testing (TODO SHOW TESTING)

Running the Server on the Cloud

http://ec2-35-166-54-53.us-west-2.compute.amazonaws.com/index.php

Installed a web server
scp to copy my index.php over to the web server’s root directory

Easy XKCD Modification (2 hours)

Comics are identifed by their number stored in the integer ComicFragment.lastComicNumber. This number is used as the comic name in the database.

Plan: In MainActivity menu, add items “Add Tag”, “Remove Tag”, and “Search Tag”
- Add Tag: create popup that takes a text input for the tag and send request to server
- Remove Tag: popup takes input for the tag and asks server to remove it from the current comic
- Search Tag: make user enter tag and then list all comics with the tag
- View Tags: list all tags associated with the current comic
- List Tags: displays all the tags that have been created for all comics

The following changes were made to the EasyXKCD APK:
- added the above menu items to menu_comic_fragment.xml
- added listener code for the new menu items in ComicFragment.java. Functions that require user input such as add, remove, or search tag use a popup menu with a text box to get a string from the user.
- added an IntentService TagQueryService which communicates to the EC2 server

- added a broadcast receiver to ComicFragment.onCreate() to listen for callbacks from TagQueryService. The receiver displays the server's response on screen using the already existing showTranscript method which creates a popup box with some text.

Known issues: The app seems to crash the first time I run the APK over ADB but then works on subsequent runs.

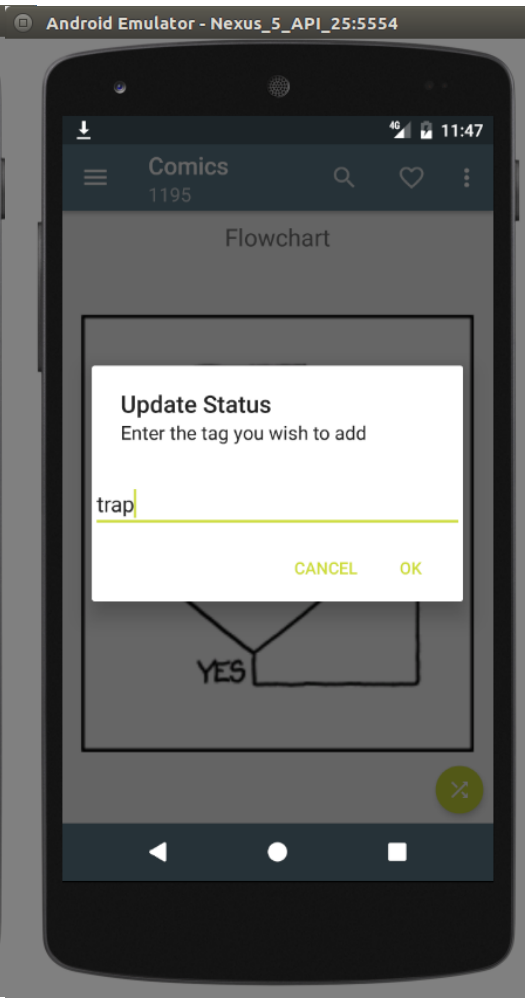
Screenshots

The five functions for tagging were added to the app’s menu as shown below.

Tagging a Comic

Viewing the Comic's Tags

Selecting 'Add Tag' Dialog



Tag Added After Pressing OK

Untagging a Comic

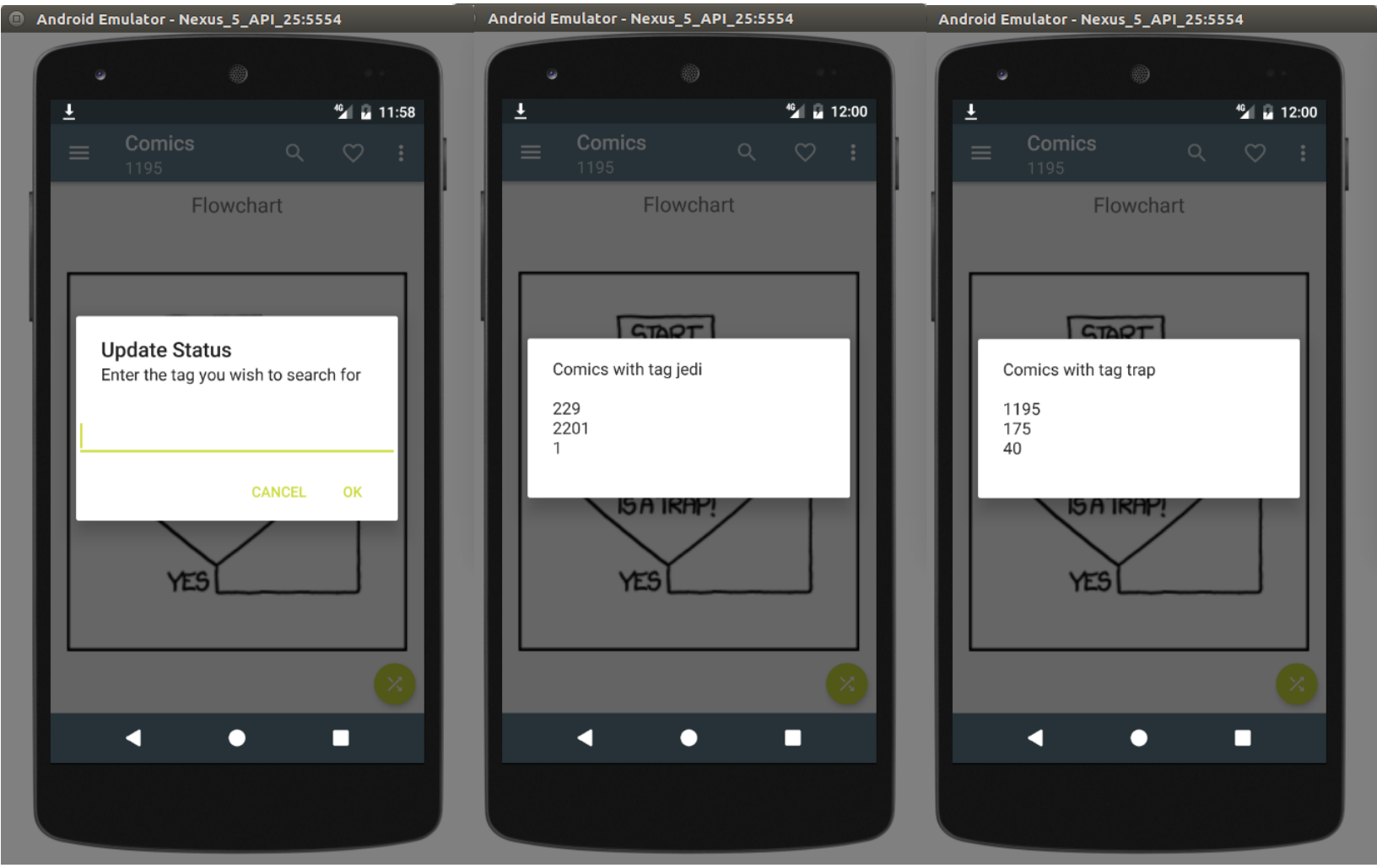
Selecting remove tag brings up a text entry dialog like the one for adding a tag. The tag entered by the user is removed from the comic (it used to have the tags graph, goats, and trap as shown above).

Viewing all tags

Selecting the ‘View Tags’ item queries the server for list of all tags that are in use. Note this image was made before the ones for ‘Add Tag’ (above) so the tags ‘trap’ and ‘graph’ do not appear in the list.

Searching for comics with a tag

The ‘Search Tag’ menu option brings up the dialog shown on the left. The center and right show the results of searching for ‘jedi’ and ‘trap’ respectively. The server returns a list of the comics with the given tag (comics are named by their number).



3.5 Analyses of Password Dumps (5 hours)

The PasswordSecure APK queries a cloud server with a password. The cloud server then checks the password against several lists and warns the client if its password is similar to one in the word lists.

Android APK (1 hour)

This is mostly a copy of the most frequent words APK from P4 with the user interface and server information changed. It contains an activity with text box/button for user input submission and a service which acts as a client for a Java server running on EC2. The server runs on port 8080 of the same EC2 instance as 3.4.

EC2 Java Server (4 hours)

File: PasswordSecurityServer.java
The server code of the server is the same as the ones from P4 with the process request method changed to implement password checking.

Password Checking

Word Lists

Several lists of words were placed on the server. The text file ‘wordlists.txt’ is used by the Java program to tell it which word lists to use. I used the word lists rockyou.txt and 10_million_password_list_top_1000000.txt.

Defining Similarity

Outputs different messages depending on the level of similarity between the password being checked and passwords from the word lists. Similarity is defined according to

BAD! If an exact match is found. The checker stops searching at this point and sens the message back to the user

WARN! If a match is found ignoring case. The checker will continue to look for exact matches

NOTE: Password may be similar to: “pass”. The beginning or ending of the password matches another password in the database (ignoring case). This will flag passwords made by prepending or appending numbers or symbols to a word. Multiple notes may be outputted per password. It is left up to the user to evaluate the level of similarity between his password and the already existing one.

One improvement that could be made is to sort the NOTEs by level of similarity. For example, ‘82password’ gives notes as similar to password (which it is) but ‘y49xV8ga9’ is noted as similar to ‘y’ which is not really accurate.

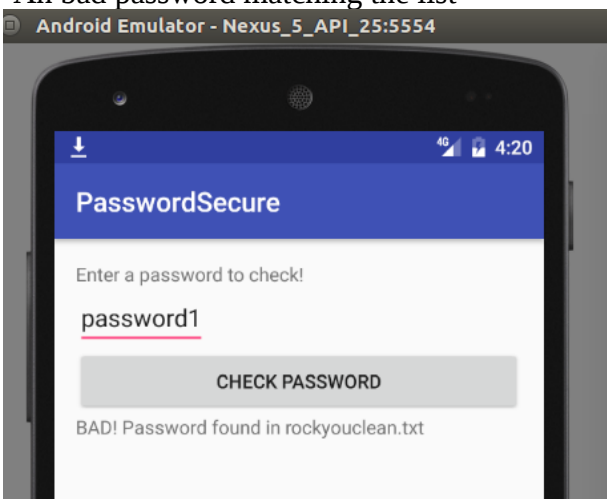
Implementation

I spent several hours tryng to track down a charset exception in my java server. It turned out that my word list (rockyou from skullsecurity) contained some bad characters. I ended up using grep, diff, and sed to find all non utf-8 words, remove those words, and place them in a new file.

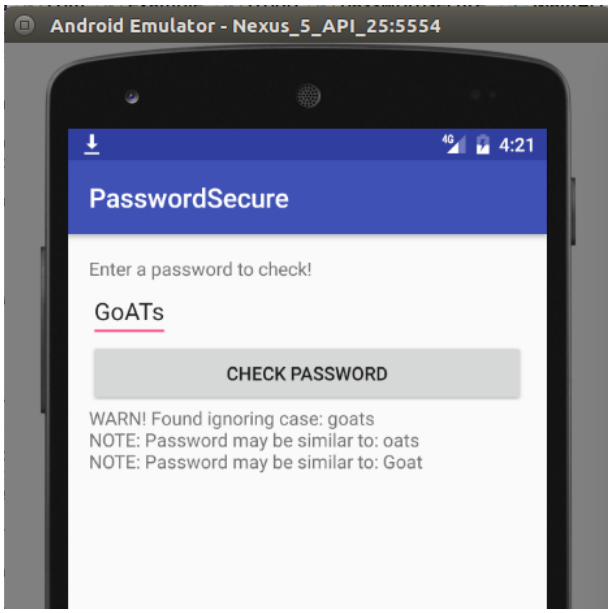
The code for the server is contained in PasswordSecurityServer.java.

Screenshots

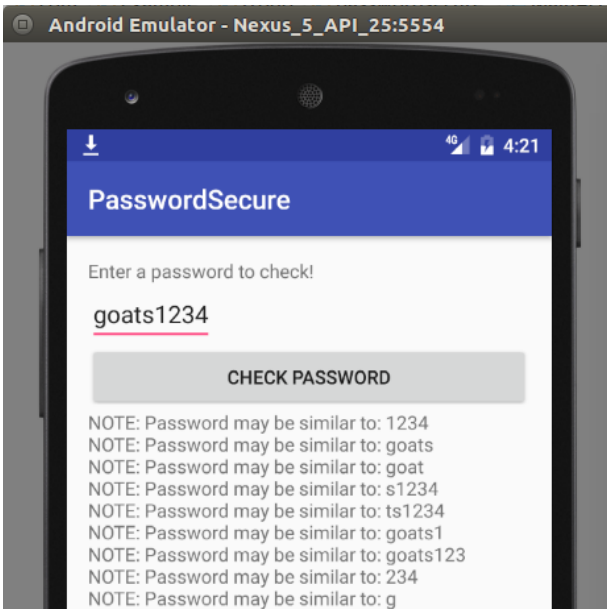
An bad password matching the list



A bad password matching ignoring case

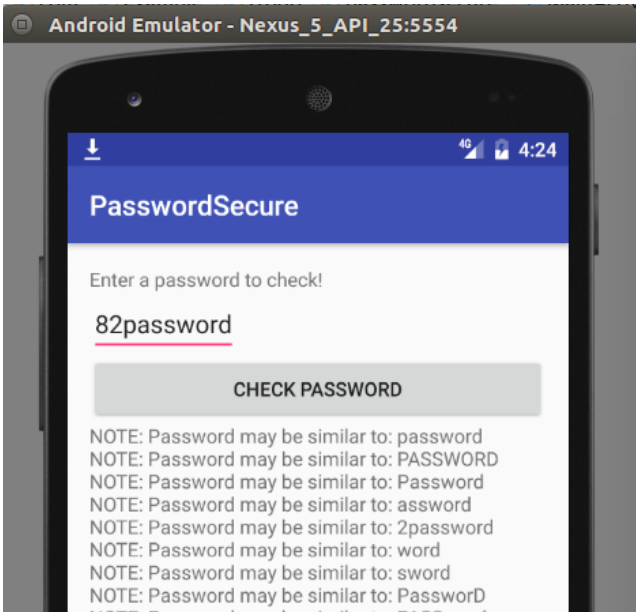


A bad password similar to others

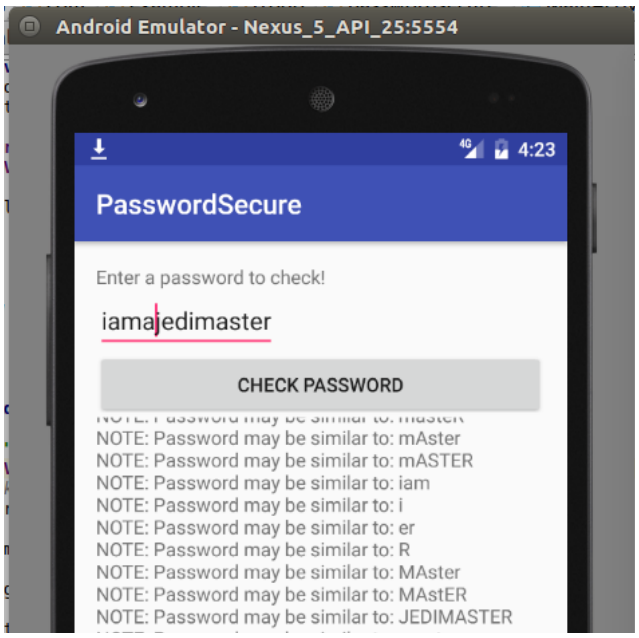


Interpretation of the severity of “NOTE: Password may be similar to: ...” messages is left to the user. Below are examples showing the notes generated for passwords of varying strengths.

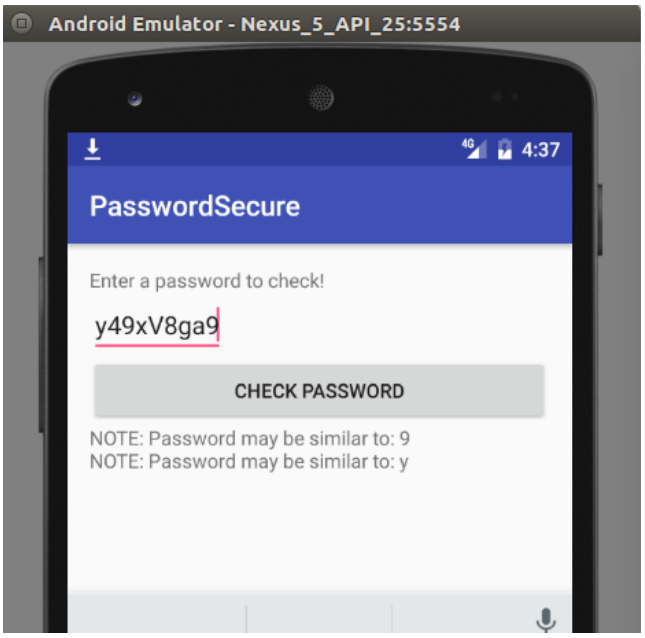
A bad password



A weak password



A perfectly fine password



Closing Thoughts

This project felt quite long. Including time to write up this project, I spent close to 30 hours on it. On the plus side, I gained a lot of new knowledge dealing with password lists, SQL, PHP, and other cloud providers, but didn’t have the time to explore some of the topics as thoroughly as I would have liked. I’m not sure what the future projects will be like, but I think it may be beneficial for them to be a little shorter and a little more focused on one aspect of cloud computing.